

Diagram illustrating a multi-stage sorting network (e.g., a merge sort implementation). The network consists of 8 input nodes (100-107) merging into 4 intermediate nodes (108-111), which then merge into 2 nodes (112, 113), and finally into a single output node (114). Each node is represented as a 3x1 grid of numbers.

Legend (Top):

addr	id	key
------	----	-----

Input Nodes (100-107):

- 100: [0, 0, 8]
- 101: [1, 1, 12]
- 102: [2, 2, 5]
- 103: [3, 3, 7]
- 104: [4, 4, 10]
- 105: [5, 5, 1]
- 106: [6, 6, 11]
- 107: [7, 7, 6]

Intermediate Nodes (108-111):

- 108: [8, 0, 8] (Merges 100, 101)
- 109: [9, 2, 5] (Merges 102, 103)
- 110: [10, 5, 1] (Merges 104, 105)
- 111: [11, 7, 6] (Merges 106, 107)

Final Merging Stage:

- 112: [12, 2, 5] (Merges 108, 109)
- 113: [13, 5, 1] (Merges 110, 111)
- 114: [14, 5, 1] (Merges 112, 113)

	id	key
0	0	8
1	1	12
2	2	5
3	3	7
4	4	10
5	5	1
6	6	11
7	7	6
8	0	8
9	2	5
10	5	1
11	7	6
12	2	5
13	5	1
14	5	1

Figure 2

Figure 1 is a diagram illustrating a hierarchical tree structure, likely representing a data storage or processing flow. The tree consists of four levels of nodes, labeled 100 through 114.

- Level 0 (Input):** Eight nodes, each containing three values: **addr**, **id**, and **key**.
  - Node 100: [0 | 0 | 8]
  - Node 101: [1 | 1 | 12]
  - Node 102: [2 | 2 | 5]
  - Node 103: [3 | 3 | 7]
  - Node 104: [4 | 4 | 10]
  - Node 105: [5 | 5 | 9]
  - Node 106: [6 | 6 | 11]
  - Node 107: [7 | 7 | 6]
- Level 1:** Four nodes, each containing three values.
  - Node 108: [8 | 0 | 8] (receives input from 100 and 101)
  - Node 109: [9 | 2 | 5] (receives input from 102 and 103)
  - Node 110: [10 | 5 | 9] (receives input from 104 and 105)
  - Node 111: [11 | 7 | 6] (receives input from 106 and 107)
- Level 2:** Two nodes, each containing three values.
  - Node 112: [12 | 2 | 5] (receives input from 108 and 109)
  - Node 113: [13 | 7 | 6] (receives input from 110 and 111)
- Level 3 (Output):** One node containing three values.
  - Node 114: [14 | 2 | 5] (receives input from 112 and 113)

The diagram shows a clear flow from the input nodes (100-107) through intermediate nodes (108-111) to the final output node (114), with each node's internal state (addr, id, key) being updated or transformed at each step.

Figure 3

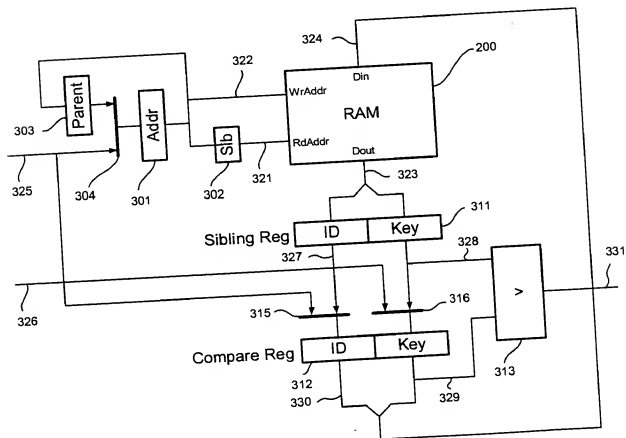


Figure 4

```
graph TD
    Start((Start)) --> 401[401 Load Compare Reg and Load Addr Reg With Changed Key and ID from input]
    401 --> 402[402 Write Compare Reg to Table Read Sibling Reg from Table Load Addr Reg with Parent Address]
    402 --> 403{403 Sibling Key < Compare Key}
    403 -- Yes --> 404[404 Load Compare Reg from Sibling Reg]
    403 -- No --> 405{405 Addr Reg Points Above Root}
    404 --> 405
    405 -- Yes --> End((End))
    405 -- No --> 402
```

The flowchart illustrates the second embodiment of the present invention. It begins with a 'Start' terminal, leading to step 401: 'Load Compare Reg and Load Addr Reg With Changed Key and ID from input'. From step 401, the flow proceeds to step 402: 'Write Compare Reg to Table Read Sibling Reg from Table Load Addr Reg with Parent Address'. Step 402 leads to decision point 403: 'Sibling Key < Compare Key'. If the answer is 'Yes', the flow goes to step 404: 'Load Compare Reg from Sibling Reg'. If the answer is 'No', the flow goes directly to decision point 405: 'Addr Reg Points Above Root'. From step 404, the flow also goes to decision point 405. If the answer to 405 is 'Yes', the flow ends at the 'End' terminal. If the answer is 'No', the flow loops back to step 402.

Figure 5

# 2025

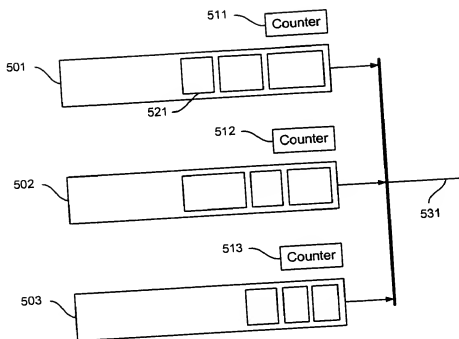


Figure 6  
Prior Art

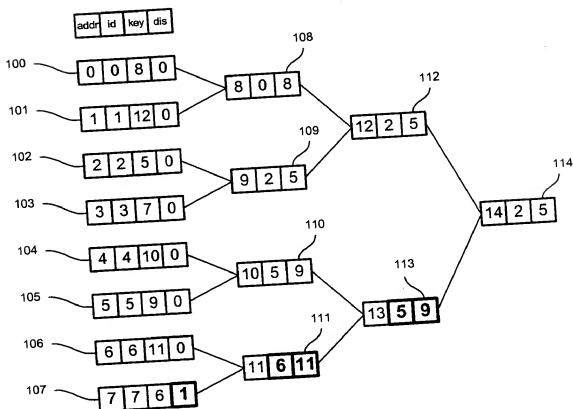


Figure 7

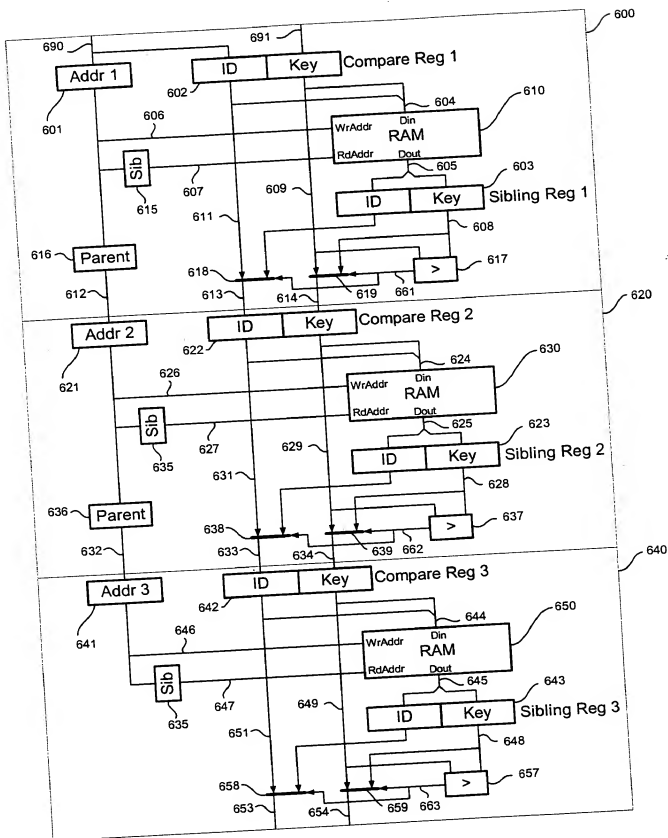


Figure 8

	Cycle				
	1	2	3	4	5
Addr 1	5	7	2		
Comp 1	5.9	7.4	2.15		
Sib 1	4.10	6.11	3.7		
Addr 2		10	11	9	
Comp 2		5.9	6.11	3.7	
Sib 2		7.6	5.9	0.8	
Addr 3			13	13	12
Comp 3			7.6	5.9	3.7
Sib 3			2.5	2.5	5.9
Result			2.5	2.5	3.7

Figure 9

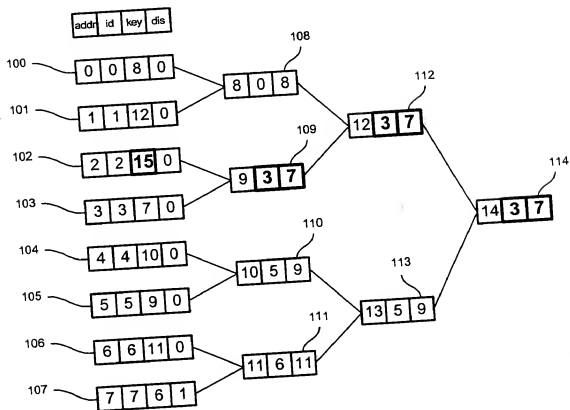
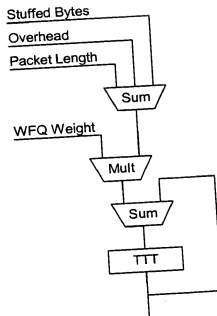
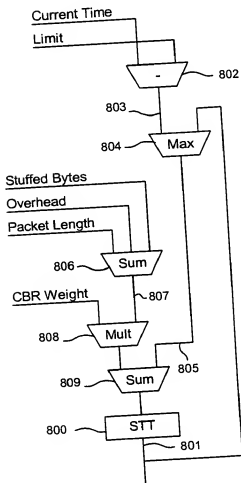


Figure 10



```

graph TD
    Start((Start)) --> 901
    901[901  
WSTT = key portion of  
CBR winner  
WCID = ID portion of CBR  
winner] --> 902
    902{902  
WSTT <= current time}
    902 -- Yes --> 903
    902 -- No --> 906
    903[903  
Initiate transmission of next  
packet from queue WCID] --> 904
    904[904  
Update counter for queue WCID  
to NSTT  
(See Figure 12)] --> 905
    905[905  
Run incremental CBR  
tournament to propagate  
change of WCID count to NSTT] --> 901
    906[906  
WTTT = key portion of WFG  
winner  
WWID = ID portion of WFG  
winner] --> 907
    907[907  
Initiate transmission of next  
packet from queue WWID] --> 908
    908[908  
Update counter for queue  
WWID to NTTT  
(See Figure 13)] --> 909
    909[909  
Run incremental WFG  
tournament to propagate  
change of WWID count to  
NTTT] --> 906

```

Figure 14



Figure 16

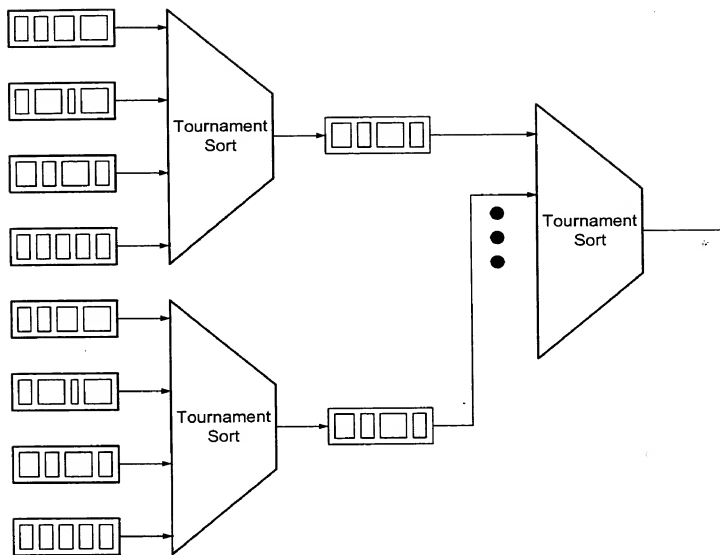


Figure 17

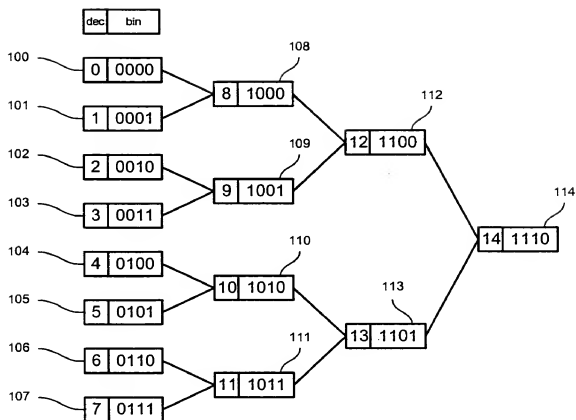


Figure 18

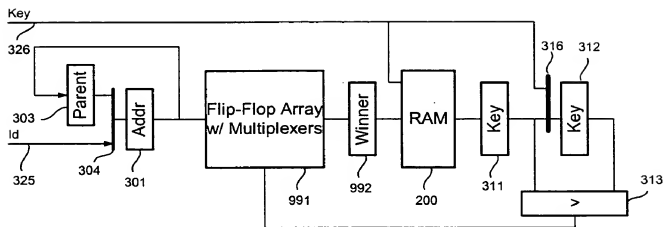


Figure 19

Figure 20